DeepLearning.AI Stanford ONLINE

SUPERVISED MACHINE LEARNING: REGRESSION AND CLASSIFICATION

MACHINE LEARNING SPECIALIZATION

DeepLearning.AI Stanford ONLINE

MACHINE LEARNING

- **Machine Learning (ML)** is the process of learning from data without explicit programming.
- Machine Learning algorithms learn from data through finding pattern and analysis
- Higher the amount of data the better is the performance
- **Types of ML**: Supervised Learning, Unsupervised Learning, Reinforcement Learning
- **Supervised Learning**: Learn from **labeled** input data, **EX**: Spam filtering, Speech recognition, Machine translation, Online advertisement, Automation
- **Regression** and **Classification** are Supervised Learning applications
- **Regression** predicts numeric value such as price of the house from size of house as input data
- **Classification** categorizes data into N small number of possible classes through decision boundary where input data is multidimensional, **EX**: Classify if image is a cat image

DeepLearning.AI Stanford ONLINE

MACHINE LEARNING

- Unsupervised Learning: Identify pattern, insight and structure from unlabeled input data. EX: Clustering, Anomaly Detection, Dimensionality Reduction, Market Segmentation
- Clustering groups unlabeled data into clusters based on properties, EX: Google News, DNA Gene Micro Array, Customer Grouping
- Anomaly Detection finds fraud transaction in credit card, fault in manufacture components
- Dimensionality Reduction compress the data for easy store and process for analysis
- Market Segmentation groups similar customers together based on property
- Linear Regression model fit straight line to estimate numeric value such as housing price
- Classification model predict category from N possible classes such as disease classification
- Input data X is called feature and output data Y is called target variable
- X fed into learning algorithm predicts y based on Hypothesis F(X)

LINEAR REGRESSION

- Linear Regression Model or **Hypothesis** fits a straight line function

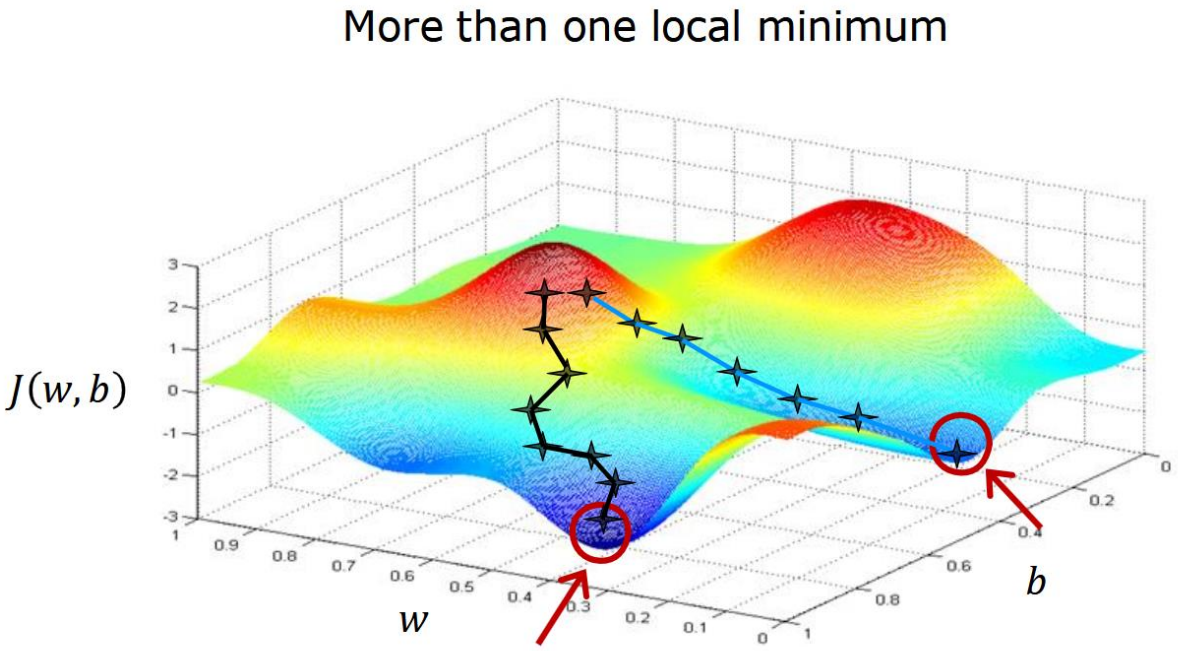
$$f_{(w,b)}(X) = wX + b$$

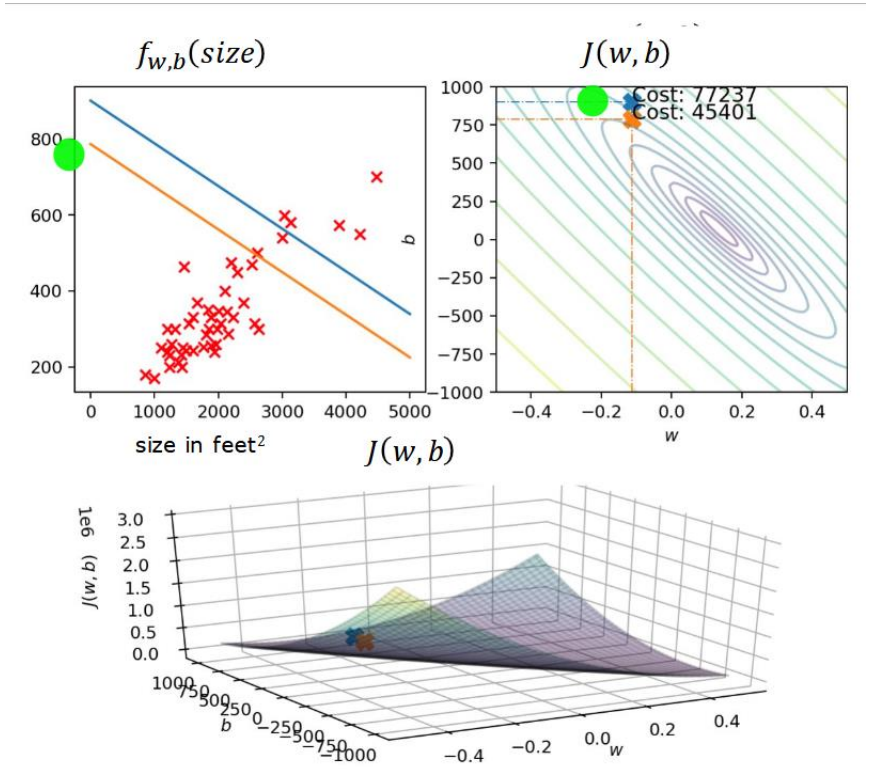
  where 'X' is the input training data, 'w' is the weight parameter and 'b' is the bias constant

- Linear Regression Model running on one variable input data is **Univariate Linear Regression**
- **Training dataset 'X' has total 'm' data points** where i^{th} data point denoted as (x^i, y^i)
- Linear Regression Model optimization finds weight 'w' and bias 'b' value that yields hypothesis value $\hat{y} = f_{(w,b)}(X) = wX + b$ close to original output 'Y'.
- **Cost Function (Sum of Squared Error):** $J_{(w,b)} = \frac{\Sigma(\hat{y} - y)^2}{2m}$ calculates the prediction error
- Goal of Linear Regression Model is to minimize error *J(w,b)* which means finding 'w' and 'b' that reaches **global minima** of the squared error cost function (typically **convex function**)
- Cost Functions are chosen such that it is **convex** or has only **one global minima**

LINEAR REGRESSION

- Contour Plot shows the cost function J in terms of 'w' and 'b' to visualize minimum error
- Contour Plot is not feasible for linear regression with higher number of 'w' parameter





GRADIENT DESCENT

- Gradient Descent: Starting with ‘w’ and ‘b’ value zero, the optimization algorithm iteratively minimize cost function J to reach the global minima of cost function. Cost function can have multiple local minima which leads different path to valley from hill. Each step updates ‘w’ and ‘b’ with cost function derivative which works as slope or direction (left or right) to reach minima from hill. Run Gradient Descent until weight parameters converge or does not update much.
- Cost Function Derivates:

$$w = w - \alpha \frac{\partial}{\partial w} J_{(w,b)} = w - \frac{\alpha}{m} \Sigma \left(f^{i}_{(w,b)} - y^{i} \right) x^{i}$$

$$b = b - \alpha \frac{\partial}{\partial b} J_{(w,b)} = b - \frac{\alpha}{m} \Sigma \left(f^{i}_{(w,b)} - y^{i} \right)$$

- α is the learning rate provides the fixed step size while climbing down from hill
- Very small learning rate α needs more iteration to reach minimum J
- Very large learning rate α overshoot minima or diverge from minima
- Batch Gradient Descent: Every step of iteration utilizes all input data points
- Mini Batch Gradient Descent: Every step of iteration utilizes small amount of input data

MULTIPLE LINEAR REGRESSION

- Multiple Linear Regression performs regression on input data with multiple parameters or features which means the input data is multi dimensional
- $x_j = j^{th}$ feature of input data X, $x^i = i^{th}$ example of input data X, n = # of features, X = [m,n]
- Multiple Linear Regression Model Hypothesis:

$$f_{(w,b)}(X) = w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n + b = np.dot(x, w) + b$$

- Python numpy vectorization perform addition and multiplication in parallel
- Vectorization is efficient as it scales to large data calculation as follows

$$X = [m, n], Y = [m, 1], W = [n, 1], b$$

$$Hypothesis:\ f_{wb} = np.dot(X, W) + b, \qquad Cost: J_{wb} = \frac{np.sum\left((f_{wb} - Y)^2\right)}{2m}$$

$$Derivates: dj_{db} = \frac{np.sum(f_{wb} - Y)}{m}, dj_{wb} = \frac{np.dot\left(X.T, (f_{wb} - Y)\right)}{m}$$

FEATURE SCALING

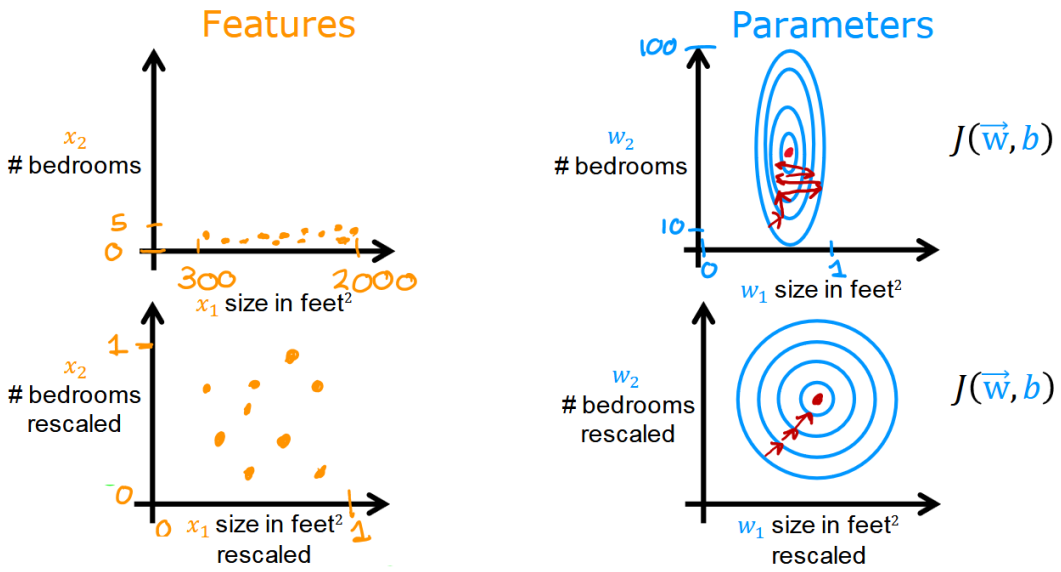
- Normal Equation: Closed form equation to solve the weights of Linear Regression

$$Weights:\ W = (X^T X)^{-1}.(X^T Y), \qquad Cost:\ f_{wb} = \frac{(XW - Y)^T (XW - Y)}{2m}$$

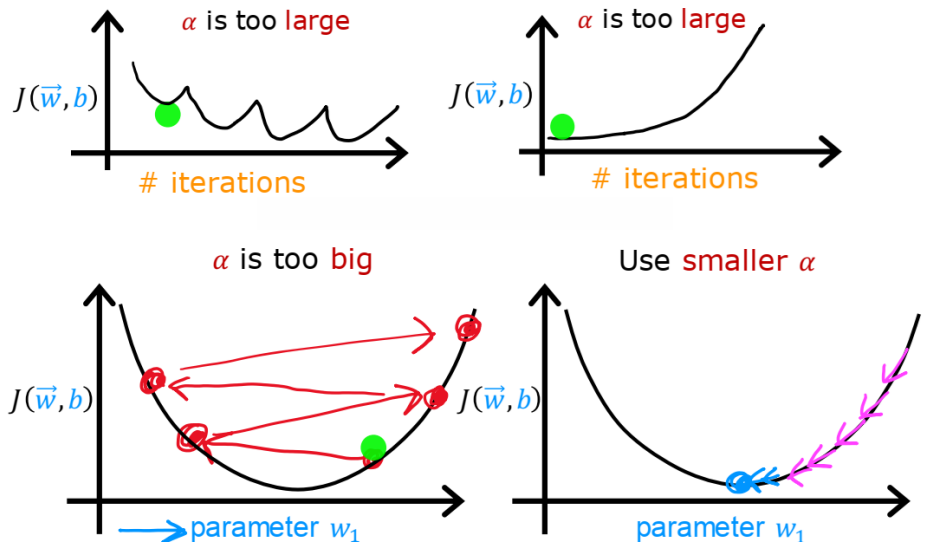
- Normal Equation solution runs slow when number of feature n is large
- Feature Scaling: Scale features to plot uniformly in specific or comparable range which makes the contour plot of features a circle. Scaling prevents feature bias and helps converge faster. Scale large feature values comparable to other smaller features.
- Scale feature with max value yield $0 < \frac{x}{maxValue} < 1$, acceptable range [-3, 3]
- Mean Normalization: Scale features $\frac{x - \mu}{\max - min}$ to center around zero with in range -1 and +1
- Z-score Normalization: Scale features $\frac{x - \mu}{\sigma}$ to center around zero with in range -3 and +3

DeepLearning.AI Stanford ONLINE

LINEAR REGRESSION



Feature Scaling



Learning Rate

LINEAR REGRESSION

- **Gradient Descent Convergence:** Learning Curve plot of cost J in terms of iteration shows cost J decreasing over iteration.
- **Auto converge** test can be performed by check decrease of cost J less than constant $\epsilon = 0.001$
- **Learning curve** creates wiggle or diverge or increase of cost J if learning rate α is very large
- Choose values of learning rate α from **0.001** to 1 increasing by 3 fold as 0.001, 0.003, 0.01, 0.03, 0.1
- **Feature Engineering**: Choose custom features by **transforming or combining** original features

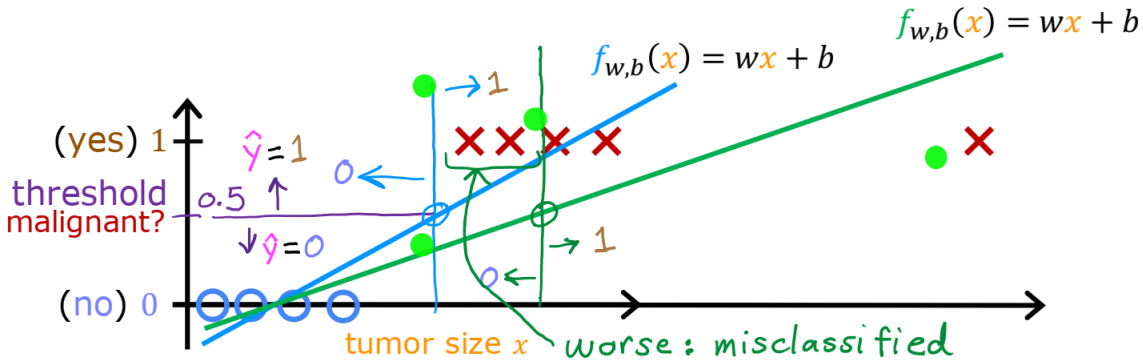
  EX: Take area as a new feature for predicting housing price from given input length and width
- **Polynomial Regression**: Polynomial function of 2nd, 3rd, 4th or 5th degree is chosen as hypothesis to **fit curve or non-linear function** for the input data

$$f_{(w,b)}(X) = w_1x_1 + w_2{x_2}^2 + w_3{x_3}^3 + b \text{ or } f_{(w,b)}(X) = w_1x_1 + w_2\sqrt{x_2} + b$$

- Feature scaling is needed for **polynomial regression** to keep features in comparable range

LOGISTIC REGRESSION

- Logistic Regression solves classification problem by predicting output category from a group of classes. Categorization from only two output classes (Y/N, T/F) is Binary Classification.
- Decision Boundary separates class points, can be linear or non-linear lines
- Linear Regression function with threshold in straight line decision boundary misclassify new value as the best fit line shift to the left or right for new data.
- Logistic Function classifies regression model output or fit into classes based on threshold

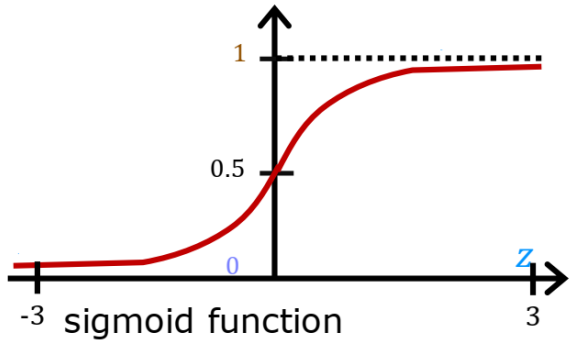


if $f_{w,b}(x) < 0.5 \rightarrow \hat{y} = 0$

if $f_{w,b}(x) \geq 0.5 \rightarrow \hat{y} = 1$

DeepLearning.AI Stanford ONLINE

LOGISTIC REGRESSION

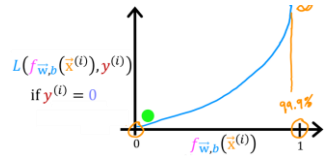


- Sigmoid Function output between 0 and 1, threshold can be 0.5
- Interpretation: $f_{wb}(x) = P(y = 1|x, w, b)$
- Linear Decision Boundary: $f_{(w,b)}(X) = w_1x_1 + w_2x_2 + b$
- Non-Linear Decision Boundary:

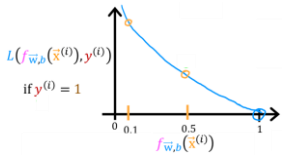
$$g(z) = \frac{1}{1+e^{-z}} \quad 0 < g(z) < 1$$

$$f_{\vec{w},b}(\vec{x}) = g(\vec{w} \cdot \vec{x} + b) = \frac{1}{1 + e^{-(\vec{w}\cdot\vec{x}+b)}}$$

$$f_{(w,b)}(X) = w_1{x_1}^2 + w_2{x_2}^3 + b$$
$$f_{(w,b)}(X) = w_1x_1 + w_2x_2 + w_3{x_1}^2 + w_4{x_2}^2 + w_5x_1x_2 + b$$



- Logistic Regression hypothesis feds Linear hypothesis into Sigmoid Function as output [0,1]
- Cost Function: Squared error cost function with Sigmoid function input creates wiggle non-convex curve with lots of local minima. Log loss function is used to predict true Y label with –log(f_wb (X)) function and false Y label with –log(1–f_wb (X)) function where loss is close to zero as Y=1 and Y=0.



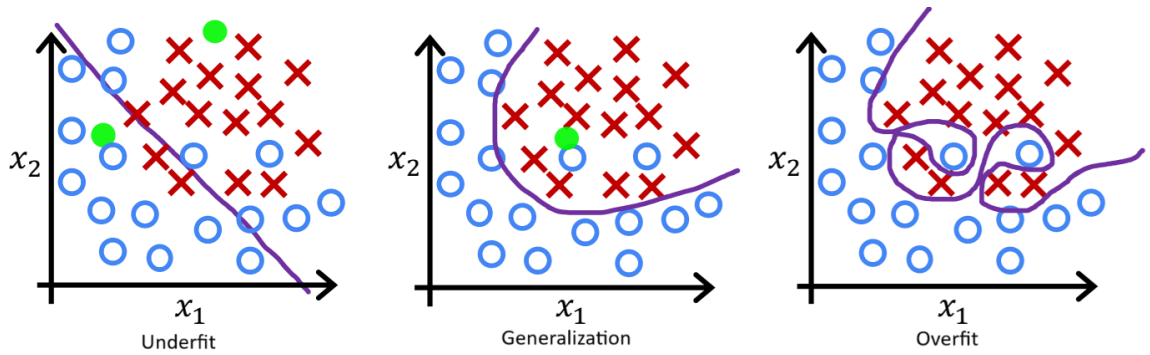
$$J_{wb} = \frac{1}{m}\sum L\left(f_{wb}\left(x^i\right), y^i\right) = -\frac{1}{m}\sum\left(y^i \log\left(f_{wb}\left(x^i\right)\right) + \left(1 - y^i\right)\log\left(1 - f_{wb}\left(x^i\right)\right)\right)$$

OVERFIT & UNDERFIT

- **Gradient Descent:** Logistic Regression weight gradients similar to Linear Regression

$$w = w - \alpha \frac{\partial}{\partial w} J_{(w,b)} = w - \frac{\alpha}{m} \Sigma (f^i_{(w,b)} - y^i) x^i \qquad b = b - \alpha \frac{\partial}{\partial b} J_{(w,b)} = b - \frac{\alpha}{m} \Sigma (f^i_{(w,b)} - y^i)$$

- **Underfit:** Model does not fit well in training data, low training accuracy, high loss and high bias, poor decision boundary and simple classification
- **Generalization:** Good prediction with high training and test accuracy
- **Overfit:** Model fits training set very well but fails in test set, high variance (polynomial function), extreme complex decision boundary, over classification. High variable in prediction for small changes in training set. **Reason:** Too many features and low data



REGULARIZATION

- Address Overfit: Collect more training data, Select features only that impact output (chances of loosing useful feature), Regularization (Large number of features)
- Regularization: Reduce the impact or effect of some features in hypothesis by applying smaller weight values which is less likely to overfit. Update cost function by adding weight parameters to minimize weight values. Regularization needed when number of feature is large (100~1000).
- Regularized Cost Function: $J_{(w,b)} = \frac{\sum(f_{wb}(x)-y)^2}{2m} + \frac{\lambda}{2m}\sum w_j^2$ (Regularization Term)
- λ is the regularization parameter which keep balance between fitting data and reducing overfit
- λ is too small then overfit and λ is too large then underfit
- Logistic and Linear Regression cost function derivative for weights with regularization

$$\frac{\partial}{\partial w_j} J_{(w,b)} = \frac{1}{m}\sum\left(f^i_{(w,b)} - y^i\right)x^i + \frac{\lambda}{m}w_j$$

- Weight w_j shrink with regularization by $(1 - \frac{\alpha\lambda}{m})$ in each iteration.